DS 1

le 27/09/2025 – Durée : 2h

- L'usage de la calculatrice n'est pas autorisé.
- La clarté et la précision des raisonnements interviendront pour une grande part dans la notation. La présentation (en particulier les indentations) est également essentielle.
- Le résultat d'une question peut être admis afin de traiter une question suivante ; une fonction peut être utilisée dans la suite d'un exercice même si elle n'a pas été écrite.
- L'exercice 3 est à aborder en dernier et sera compté en bonus.

Exercice 1

L'objectif de cet exercice est d'étudier une méthode générale d'intégration. On l'utilisera pour obtenir une valeur numérique approchée de

$$I = \int_0^\pi \frac{\cos^2\left(\frac{\pi}{2}\cos\theta\right)}{\sin\theta} d\theta \ .$$

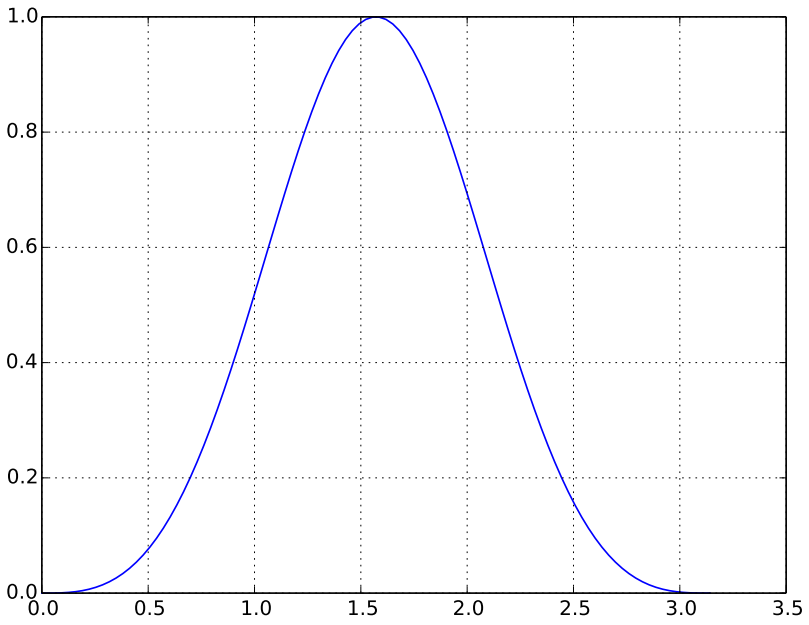
L'application $f : x \mapsto \dfrac{\cos^2\left(\frac{\pi}{2}\cos x\right)}{\sin x}$ est continue sur $]0;\pi[$ et prolongeable par continuité en 0 et π en posant $f(0) = f(\pi) = 0$. L'intégrale I est donc bien définie.
La méthode des rectangles consiste à approcher l'intégrale par la suite $(R_n)_{n\in\mathbb{N}^*}$ définie pour notre fonction f par :

$$R_1 = 0 \qquad \text{et pour tout } n \geq 2,\ R_n = \lambda_n \sum_{k=1}^{n-1} f\left(\frac{k\pi}{n}\right) \ ;$$

où λ_n désigne un nombre réel qu'il faudra préciser.

On n'utilisera dans ce qui suit aucune bibliothèque python, mais on considèrera que les fonctions `cos`, `sin` et le nombre `pi` ont été importés.

Question 1 – Illustrer le principe de la méthode des rectangles en représentant R_{12} sur le graphe de la fonction f (vous pouvez rendre l'énoncé complété).
Donner l'expression de λ_n en fonction de n.



Question 2 – Écrire une fonction `f(x)` qui retourne la valeur de f pour un réel $x \in]0;\pi[$.

Question 3 – Écrire une fonction `rectangle1(n)` qui, pour un nombre entier $n \geq 1$, renvoie la valeur de R_n.

Question 4 – Combien d'évaluations de la fonction f nécessite cette fonction pour une valeur de n donnée ?

On souhaite déterminer une valeur de n pour laquelle R_n fournit une bonne estimation de I. La méthode 1 consiste à calculer les termes successifs de la suite $(R_n)_{n\geq 1}$ jusqu'à ce que celle-ci semble se stabiliser.

Question 5 – Compléter la fonction `integrale1(eps)` qui calcule, pour $eps > 0$, les termes successifs de la suite $(R_n)_{n\geq 1}$ jusqu'à ce que $|R_n - R_{n-1}| < eps$, puis retourne la dernière valeur de R_n calculée. La fonction valeur absolue est obtenue par `abs`.

```
def integrale1(eps):
    n = 2
    R_prec = rectangle1(n-1) # R_{n-1}
    R = rectangle1(n) # R_n
    while ...           >= eps :
        ...
        R_prec = ...
        R = ...
    return R
```

Question 6 – Pour $eps = 10^{-7}$, la fonction `integrale1(eps)` s'arrête pour $n = 34$. Vérifier que cela a nécessité plus de 500 évaluations de la fonction `f`.

Il est possible d'améliorer la méthode 1. La méthode 2 repose sur la relation de récurrence suivante :

$$\forall n \in \mathbb{N}^*, \quad R_{2n} = \frac{R_n}{2} + \frac{\pi}{2n} \sum_{k=1}^{n} f\left(\frac{(k-1/2)\,\pi}{n}\right) . \qquad (1)$$

Question 7 – Écrire la relation de récurrence pour un entier n de la forme $n = 2^{m-1}$.
Écrire une fonction récursive `rectangle2(m)` qui pour un nombre entier m retourne R_{2^m}.

Question 8 – Combien d'évaluations de la fonction f nécessite cette fonction pour une valeur de m donnée ?

Question 9 – Écrire une fonction `integrale2(eps)` qui calcule, pour $eps > 0$, les termes successifs de la suite R_{2^m} jusqu'à ce que $|R_{2^m} - R_{2^{m-1}}| < eps$, puis retourne la dernière valeur de R_{2^m} calculée.
Cet algorithme doit être élaboré dans le but de minimiser le nombre total d'évaluations de f.

Question 10 – Pour $eps = 10^{-7}$, la fonction `integrale2(eps)` s'arrête pour $m = 7$. Combien d'évaluations de la fonction f ont été nécessaires ?

Question 11 – Démontrer la formule (1).

Exercice 2

Cet exercice aborde divers aspects de la recherche du nombre d'éléments distincts dans une liste. Les trois parties sont largement indépendantes.

1. **Une première version de la recherche du nombre d'éléments distincts**

   (a) Écrire une fonction `Premier(L,i)` qui teste si `L[i]` apparaît pour la première fois en i-ème position dans la liste `L` (dans ce cas la fonction renverra `True`) ou si la valeur `L[i]` apparaissait déjà dans `L[:i]` (dans ce cas la fonction renverra `False`).

   (b) Écrire une fonction `NombreDistincts(L)` qui renvoie le nombre d'éléments distincts dans `L` (on pourra utiliser la fonction précédente).

   (c) Quelle est la complexité de la fonction précédente en fonction de la longueur n de `L` ?

2. **Cas d'une liste triée**
   On suppose dans cette question la liste `L` non vide et triée en ordre croissant.

   (a) Compléter la fonction `NombreDistinctsTri(L)` afin qu'elle renvoie le nombre d'éléments distincts dans `L`.

```
def NombreDistinctsTri(L):
    n=len(L)
    i=1
    N=1
    while i < n :
        if ...           :
            N = N+1
        i = ...
    return N
```

   (b) Quelle est la complexité de la fonction précédente ?

   (c) Justifier la correction de la fonction précédente (à l'aide d'un invariant).

   (d) Écrire une fonction récursive `NombreDistinctsRec(L)` qui, en s'appelant sur un tableau de taille inférieure, renvoie le nombre d'éléments distincts dans `L`. On s'autorisera à faire des copies de sous-tableaux de `L`.

   (e) On souhaite améliorer la version précédente en écrivant une fonction `NombreDistinctsRec2(L)` qui ne nécessitera plus de copie de sous-tableau. Pour cela, on s'appuie sur une fonction auxiliaire (récursive) `NombreDistinctsRecPart(L,i)` qui renvoie le nombre d'éléments distincts dans `L[:i]`.

   Compléter la fonction suivante :

```
def NombreDistinctsRec2(L):
    def NombreDistinctsRecAux(L,i):
        if i == 0:
            return ...
        elif i == 1:
            return ...
        else :
            if .. :
                return NombreDistinctsRecAux(L,i-1) + 1
            else :
                return ...

    return NombreDistinctsRecAux(... , ... )
```

3. **Tri d'une liste contenant au plus N éléments distincts connus**
   On a vu à la question précédente que le tri d'une liste rendait plus rapide le comptage de son nombre d'éléments distincts. Réciproquement, on va voir que le fait de connaître le nombre d'éléments distincts dans une liste permet d'améliorer la complexité du tri.
   Pour simplifier les choses, on suppose que les éléments de `L` sont des entiers compris entre 0 (inclus) et N (exclus). Cela peut s'appliquer lorsque les nombres sont des notes par exemple.

   (a) Écrire une fonction `Comptage(L,N)` qui renvoie une liste `C` telle que `C[k]` est le nombre d'occurences de `k` dans `L`.

   (b) Utiliser la fonction précédente pour écrire une fonction `Tri(L,N)` renvoyant une liste `M` contenant les éléments de `L` triés dans l'ordre croisant.

   (c) Quelle est, en fonction de la taille n de la liste L, la complexité de cette fonction `Tri` ? Comparer avec les tris insertion et sélection.

Exercice 3

Dans cet exercice, on s'intéresse aux points fixes des fonctions $f : E \to E$, où E est un ensemble fini. Le calcul effectif et efficace des points fixes de telles fonctions est un problème récurrent en informatique (transformation d'automates, vérification automatique de programmes, algorithmique des graphes, etc), et admet différentes approches selon la structure de E et les propriétés de f. On suppose par la suite un entier strictement positif n fixé et rangé dans une constante globale de même nom, et on pose $E_n = \{0, \dots, n-1\}$. On représente une fonction $f : E_n \to E_n$ par une liste `t` de taille n, autrement dit $f(x) =$`t[x]` pour tout $x \in E_n$. Ainsi la fonction f_0 qui à $x \in E_{10}$ associe $2x + 1$ modulo 10 est-elle représentée par la liste : `[1,3,5,7,9,1,3,5,7,9]`.
On rappelle que x est un point fixe de la fonction f si et seulement si $f(x) = x$.
On notera f^k l'itérée k-ème de f ($f \circ \cdots \circ f$ k fois).

1. Écrire une fonction `points_fixes(t)` qui prend en argument une liste `t` de taille n et renvoie la liste des points fixes de la fonction $f : E_n \to E_n$ représentée par `t` (liste vide si f n'admet pas de point fixe). Par exemple, `points_fixes` devra renvoyer `[9]` pour le tableau donné en introduction, puisque 9 est l'unique point fixe de la fonction f_0.

2. Écrire une fonction `itere(t,x,k)` qui prend en premier argument une liste `t` de taille n représentant une fonction $f : E_n \to E_n$, en deuxième $x \in E_n$ et en troisième argument un entier $k \geq 0$, et renvoie $f^k(x)$.

3. Écrire une procédure `nb_points_fixes_iteres(t,k)` qui prend en premier argument une liste `t` de taille n représentant une fonction $f : E_n \to E_n$, en deuxième argument un entier $k \geq 0$, et renvoie le nombre de points fixes de f^k.

4. Un élément $z \in E_n$ est dit attracteur principal de $f : E_n \to E_n$ si et seulement si z est un point fixe de f, et pour tout $x \in E_n$, il existe un entier $k \geq 0$ tel que $f^k(x) = z$. Afin d'illustrer cette notion, on pourra vérifier que la fonction f_1 représentée par la liste `[5,5,2,2,0,2,2]` ci-dessous admet 2 comme attracteur principal. En revanche, on notera que la fonction f_0 donnée en introduction n'admet pas d'attracteur principal, puisque ${f_0}^k(0) \neq 9$ quel que soit l'entier $k \geq 0$.

   Écrire une fonction `admet_attracteur_principal(t)` qui prend en argument un tableau `t` de taille n et renvoie `True` si et seulement si la fonction $f : E_n \to E_n$ représentée par `t` admet un attracteur principal, `False` sinon.